Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação

ÉRICSEN RODRIGUES LUCAS

MÉTODOS ÁGEIS NO DESENVOLVIMENTO DE SISTEMAS DE CRÉDITO: UM RELATO DE EXPERIÊNCIA

Belo Horizonte
2016

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação
Especialização em Informática: Ênfase: Engenharia de Software

MÉTODOS ÁGEIS NO DESENVOLVIMENTO DE SISTEMAS DE CRÉDITO: UM RELATO DE EXPERIÊNCIA

por

ÉRICSEN RODRIGUES LUCAS

Monografia de Final de Curso
CEI-ES-T20-2016

Prof. Marco Túlio de Oliveira Valente
Orientador

Belo Horizonte
2016

ÉRICSEN RODRIGUES LUCAS

MÉTODOS ÁGEIS NO DESENVOLVIMENTO DE SISTEMAS DE CRÉDITO: UM RELATO DE EXPERIÊNCIA

Monografia apresentada ao Curso de Especialização em Informática do Departamento de Ciências Exatas da Universidade Federal de Minas Gerais, como requisito parcial para a obtenção do grau de Especialista em Informática.

Área de concentração: Ênfase em Engenharia de Software.

Orientador: Prof. Marco Túlio de Oliveira Valente

Belo Horizonte
2016

Ficha catalográfica elaborada pela Biblioteca do ICEx - UFMG

Lucas, Éricsen Rodrigues.

L933m Métodos ágeis no desenvolvimento de sistemas de crédito: um relato de experiência / Éricsen Rodrigues Lucas. — Belo Horizonte, 2016.
x, 53 f.: il.; 29 cm.

Monografia (especialização) - Universidade Federal de Minas Gerais – Departamento de Ciência da Computação.

Orientador: Marco Túlio de Oliveira Valente.

1. Computação 2. Engenharia de software. 3. Software - Desenvolvimento. I. Orientador. II. Título.

CDU 519.6*32(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS

INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
ESPECIALIZAÇÃO EM INFORMÁTICA: ÁREA DE CONCENTRAÇÃO ENGENHARIA DE SOFTWARE

MÉTODOS ÁGEIS NO DESENVOLVIMENTO DE SISTEMAS DE CRÉDITO: UM RELATO DE EXPERIÊNCIA

ÉRICSEN RODRIGUES LUCAS

Monografia apresentada aos Senhores:

Prof. Marco Túlio de Oliveira Valente
Orientador
DCC - ICEx - UFMG

Profa. Mariza Andrade da Silva Bigonha
DCC - ICEx - UFMG

Belo Horizonte, 15 de abril de 2016

DEDICATÓRIA

Este trabalho é dedicado às pessoas que tem caminhado junto comigo, em especial aos meus pais Vânia e Mauricio, que investiram em mim.

AGRADECIMENTOS

Agradeço primeiramente a Deus, que me ouviu nos momentos difíceis, me consolou, me proporcionou a capacidade de seguir em frente e chegar onde eu estou.

Agradeço aos meus pais Vânia e Mauricio pelo apoio incondicional em meus sonhos, pelos investimentos feitos em mim. Assim como todo carinho, educação, dedicação, amor e confiança depositada.

Agradeço a esta universidade e seu corpo docente que oportunizaram a janela que hoje vislumbro um horizonte superior.

Agradeço ao meu orientador Marco Túlio, pelo suporte e correções.

E a todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

EPÍGRAFE

"Tudo tem o seu tempo determinado, e há tempo para todo propósito debaixo do céu."

Eclesiastes 3:1

RESUMO

Com a necessidade de desenvolver softwares em ambientes turbulentos dos negócios, onde os requisitos mudam constantemente os métodos ágeis surgiram. O propósito dos métodos ágeis é reduzir as burocracias dos processos tradicionais de desenvolvimento de software, focando no desenvolvimento, de forma simples, rápida e sem documentações extensas evitando redundância, excesso e auxiliando efetivamente no desenvolvimento de software.

O presente trabalho teve como objetivo identificar a metodologia que melhor se adequa no desenvolvimento de sistemas de concessão de crédito. Foram realizados estudos bibliográficos das metodologias ágeis *Scrum*, *Extreme Programming* e Crystal, levantando as principais características que condiz com o ambiente que seria empregado. Após a análise concluiu-se que o *Scrum* é a metodologia que melhor se aplicaria no cenário que requer soluções rápidas e interativas. Como resultado foi realizado o desenvolvimento parcial de um módulo do sistema de crediário, para dar suporte na análise das compras realizadas via do sistema de crediário, com o objetivo de reduzir o número de fraudes.

Palavras-Chave: Metodologias Ágeis, Scrum, Extreme Programming, Sistema de Crédito

ABSTRACT

With the need of software development in business troubled environments, where the requirements change constantly , the agile methods were created with the purpose of reduce the bureaucracy of the traditional process of software development, focusing in the development, in a simple way, quick and without extensive documentation avoiding redundancy, excess and helping effectively the software development.

This present work had as goal identify the methodology that better fits in the credit systems development. Were realized bibliographic studies of the methodologies Scrum, Extreme Progamming and Crystal, showing the main characteristics that would fit in the environment. After the analysis, was conclued that the Scrum would be the better choice for an environment that needs interactive and quick solutions. As the result was made the partial development of a module of the credit system which will give support the procurement system made through the credit system with the objective of reduce the number of fraud.

Keywords: *Agile Methodologies, Extreme Programming, Scrum, Credit System*

LISTA DE FIGURAS

LISTA DE SIGLAS

SCRUM: Metodologia ágil cujo foco recai sobre as atividades de gerenciamento de projetos.

XP (*Extreme Programming*): Metodologia ágil cujo foco está sobre as atividades de construção dos sistemas.

PO (*Product Owner*) – Responsável por representar os interesses de todos os envolvidos no projeto.

SUMÁRIO

1. INTRODUÇÃO

Atendendo as necessidades do mercado, as organizações estão em constante corrida para acompanhar as inovações e oferecer o melhor. Para tanto, são necessárias adaptações contínuas, pois grandes desafios são enfrentados. Desafios estes como grandes mudanças nos requisitos, necessidade de entrega constantes, exclusão de funcionalidades, código de baixa qualidade e desligamento de membros com alto grau de conhecimento, podendo causar o fracasso do projeto.

Empresas na busca por resultados mais eficientes, rapidez para lançamento de novos produtos com qualidade, baixo custo e que atenda suas necessidades em um curto espaço de tempo, estão trocando as metodologias tradicionais por uma nova gama de metodologias de desenvolvimento de software conhecidas como metodologias ágeis. Consideradas na maioria das vezes como antídoto ao excesso de burocracias e documentação extensiva, as metodologias ágeis se baseiam mais nas pessoas e suas interações, em vez de focar em grandes esforços de planejamento de requisitos e processos rígidos. Por pregar a simplicidade, acarreta em grande carga de disciplina e organização.

No contexto de agilidade e fluidez que se deu com o surgimento das metodologias ágeis para o desenvolvimento de software que surgiu metodologias como o *Scrum* do Ken Schwaber, Jeff Sutherland e Mike Beedle, *Extreme Programming* do Kent Beck, Ron Jeffries e Ward Cuningham, Crystal do Alistair Cockburn e Jim Highsmith, entre outras.

O *Scrum* conhecido como um *framework* ágil, simples e leve é utilizado para a gestão do desenvolvimento de softwares complexos em ambientes complexos, equipes pequenas, requisitos pouco estáveis e pequenas interações. Embasado no empirismo e abordagem interativa e incremental é sustentado por três pilares, sendo a transparência, inspeção e adaptação. Como objetivo, a metodologia define os processos de desenvolvimento focando nas pessoas da equipe e exige poucos artefatos.

A metodologia *Extreme Programmig* (também conhecida como XP), é voltada para equipes pequenas e médias que desenvolve softwares baseados em requisitos vagos e modificados rapidamente e é ideal para desenvolvimento incremental ou interativo. XP se destaca por *feedback* constante, abordagem incremental, encorajamento de comunicação face a face, flexibilidade na implantação de funcionalidades, dependência dos testes automatizados, programação em par, etc.

Com isto gera como resultado um conjunto de valores e práticas que, ao serem usados em conjunto, produzem um resultado estável, previsível e flexível.

1.1. OBJETIVOS

Nesta seção, serão apresentados os objetivos gerais e específicos estabelecidos para este trabalho.

1.1.1. OBJETIVO GERAL

Realizar uma pesquisa das principais metodologias ágeis, com o intuito de compreendê-las, aprendê-las e aplicar a metodologia que melhor se enquadra no desenvolvimento de um sistema de crediário.

1.1.2. OBJETIVOS ESPECÍFICOS

I. Identificar as metodologias ágeis e suas características via revisão de literatura.
II. Identificar a metodologia que melhor supre a necessidade de implantação de métodos ágeis no desenvolvimento de software de crediário.
III. Aplicar o método ágil no desenvolvimento de um projeto no sistema de crediário.

1.2. JUSTIFICATIVA

As empresas procuram cada vez mais softwares de qualidade, com baixo custo e rápidos lançamentos de novos produtos. As empresas de crediários não estão em um contexto diferente, pois precisam cada vez mais apresentar aos seus clientes um software eficiente, eficaz, seguro e inovador.

Vivenciando diariamente o desenvolvimento de um sistema de crediário onde os requisitos mudam constantemente e demandam cada vez mais rápidos lançamentos de novos módulos e versões com qualidade; a ausência de uma metodologia definida para a equipe seguir, sentimos a necessidade de estudar e identificar uma metodologia que melhor se aplica à equipe existente e condiz com a sua atual situação. Optamos pela metodologia dos modelos ágeis por considera-los mais adequados para melhorar o sistema de crediário existente. Neste contexto, nos

propusemos a aplicar a metodologia ágil mais adequada e apresentar à equipe os resultados que a mesma trouxe no desenvolvimento do projeto.

1.3. ESTRUTURA DO TRABALHO

Este trabalho está dividido, além da introdução, em seis capítulos, que estão organizados da seguinte forma:

- O capítulo 2 aborda o conceito das metodologias ágeis. O objetivo do capítulo é explicar os conceitos, como originaram-se os métodos ágeis, seus principais objetivos, caracterização e as barreiras que as organizações enfrentam com a sua adoção.
- O capítulo 3 aborda o método ágil *Scrum*. O objetivo do mesmo é apresentar a sua história, definição, seus processos, seu funcionamento, as fases que o compõe, os papéis dos envolvidos na metodologia, artefatos e as cerimônias.
- O capítulo 4 aborda o método ágil *Extreme Programming*. O objetivo do capítulo é apresentar o XP, como se originou, a sua definição, seus valores e suas práticas.
- O capítulo 5 aborda o método ágil Crystal. O objetivo do capítulo é apresentar a metodologia, dissertando sua origem, as características, a abordagem e os ciclos de vida que compões a metodologia.
- O capítulo 6 apresenta a proposta de solução para o problema definido, apresentando a implantação da metodologia *Scrum* em um projeto de um sistema de crediário.
- O capítulo 7 apresenta as considerações finais do trabalho, assim como a intenção de projetos futuros.

2. MÉTODOS ÁGEIS

Com a necessidade de adaptação contínua das organizações durante o desenvolvimento de softwares, grandes desafios são enfrentados. Boa parte desses desafios são decorrentes das constantes mudanças dos requisitos, implantações frequentes de funcionalidades no sistema e até mesmo exclusão de funcionalidade que causa grandes impactos no produto final.

Segundo Fagundes (2005) a partir da década de 90 começaram a surgir novos métodos sugerindo uma abordagem de desenvolvimento ágil, em que os processos adotados tentam-se adaptar às mudanças, apoiando a equipe de desenvolvimento em seu trabalho.

O objetivo desses métodos, segundo Highsmith e Cockburn(2001) é o de obter um desenvolvimento de software mais adequado ao ambiente turbulento dos negócios, que exige mudanças rápidas e frequentes. As metodologias ágeis surgiram como uma reação às metodologias pesadas (FOWLER,2005) e tiveram como principal motivação criar alternativas para o Modelo em Cascata (HILMAN, 2004).

Diante desse cenário em fevereiro de 2001 o termo “Metodologia Ágil” tornou-se popular. A partir da conferência conhecida como Aliança Ágil que envolvia 17 especialistas em desenvolvimento de software, realizada para discutir maneiras de melhorar o desempenho de seus projetos. Durante a conferência ficou estabelecido o Manifesto Ágil, uma declaração com os princípios que regem o desenvolvimento ágil.

Manifesto para Desenvolvimento Ágil de Software

Estamos descobrindo maneiras melhores de desenvolver software, fazendo-o nós mesmos e ajudando outros a fazerem o mesmo. Através deste trabalho, passamos a valorizar:

- **Indivíduos e interações** mais que processos e ferramentas
- **Software em funcionamento** mais que documentação abrangente
- **Colaboração com o cliente** mais que negociação de contratos
- **Responder a mudanças** mais que seguir um plano

Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda.

Kent Beck	James Grenning	Robert C. Martin
Mike Beedle	Jim Highsmith	Steve Mellor
	Andrew Hunt	Ken Schwaber

Arie van Bennekum	Ron Jeffries	Jeff Sutherland
Alistair Cockburn	Jon Kern	Dave Thomas
Ward Cunningham	Brian Marick	
Martin Fowler		

Os métodos ágeis não descartam as técnicas e também não é contra os modelos da Engenharia de Software. Segundo Highsmith (2002) os métodos ágeis incluem técnicas utilizadas na Engenharia de Software, porém não segue o padrão proposto pelas metodologias tradicionais.

De acordo com Fagundes (2005), para auxiliar as pessoas a compreender o enfoque do desenvolvimento ágil, os membros da Aliança Ágil refinaram as filosofias contidas em seu manifesto em uma coleção de doze princípios, aos quais os métodos ágeis de desenvolvimento de software devem ser adequar. Estes princípios são (Manifesto Ágil, 2001):

1. Nossa maior prioridade é satisfazer o cliente, por meio da entrega adiantada e contínua de software de valor.
2. Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adequam a mudanças, para que o cliente possa tirar vantagens competitivas.
3. Entregar software funcionando com frequência, na escala de semanas até meses, com preferência aos períodos mais curtos.
4. Pessoas relacionadas à negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto.
5. Construir projetos ao redor de indivíduos motivados. Dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho.
6. O Método mais eficiente e eficaz de transmitir informações para, e por dentro de um time de desenvolvimento, é via uma conversa cara a cara.
7. Software funcional é a medida primária de progresso.
8. Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e usuários, devem ser capazes de manter indefinidamente, passos constantes.
9. Contínua atenção à excelência técnica e bom projeto, aumenta a agilidade.

10. Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito.
11. As melhores arquiteturas, requisitos e projeto emergem de times auto-organizáveis.
12. Em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajustam e otimizam seu comportamento de acordo.

Cockburn (2000) define desenvolvimento ágil de software como uma abordagem de desenvolvimento que trata os problemas das mudanças rápidas. Um processo ágil implica em um processo adaptativo e leve, que facilmente responde a mudanças (LARMAN, 2002). As metodologias ágeis adaptam-se a novos fatores decorrentes do desenvolvimento (FOWLER, 2005) ao invés de analisar, modelar e planejar inicialmente o projeto como um todo.

Os métodos ágeis podem ser considerados como um meio termo entre a ausência de processo e o processo exagerado (ZANATTA, 2004). Optam por uma documentação apropriada evitando redundâncias e excessos, para que auxilie efetivamente o desenvolvimento do software (FILHO et alii, 2005). Aconselha a criação de documentos que têm valor (HILMAN, 2004), em que somente a documentação necessária é gerada.

Os métodos classificados como ágeis possuem em comum o fato de ser aplicado em projetos não muito complexos, utilizando ciclos iterativos curtos, planejamento guiado por funcionalidades, retro alimentação constante, tolerância a mudanças, proximidade da equipe, intimidade com o cliente e um foco no ambiente geral de trabalho do time (HIGHSMITH, 2002).

Abrahamsson e outros (Abrahamsson et. al., 2003) afirmam que métodos ágeis focam na simplicidade e na rapidez. Com isto reduz as burocracias nos processos tradicionais evitando o trabalho desnecessário, perda de tempo em funcionalidades que não atenda aos usuários e documentação que nunca serão utilizadas. No desenvolvimento do projeto considera-se as funcionalidades, realimentação e a reação rápida às mudanças tecnológicas e de negócio.

Mesmo com a crescente aceitação das equipes de desenvolvimento de software na utilização dos métodos ágeis, ainda encontra muita resistência na aceitação da cultura agilista pelas organizações. Boehm e Turner (2005) identificaram uma lista destas barreiras, mas chegaram à conclusão que grande parte delas não

são, de fato, problemas à adoção de métodos ágeis. Isto é, podem ser vencidas pelas organizações. Alguns exemplos destas barreiras são:

- garantia de segurança nos sistemas;
- agilidade é inadequada para gerência de defeitos;
- efetividade de testes automáticos na aceitação e integração de sistemas;
- percepção que agilidade é extrema ou uma moda, não responsável;
- projetos ágeis são “ingerenciáveis”.

Após o estabelecimento do Manifesto Ágil, metodologias como SCRUM, eXtreme Programming (XP), CRYSTAL, Adaptive Software Development (ASD) e entre outras começaram a ter cada vez mais adeptos. As metodologias ágeis possuem métodos, práticas e técnicas que podem aumentar a satisfação do cliente (Boehn e Turner, 2003), além de produzir um sistema com maior qualidade e em menor tempo (Anderson, 2003). Nas seções seguintes ficará claro a diferença, métodos, práticas e técnicas das principais metodologias.

3. SCRUM

Seguindo os princípios que posteriormente foram definidos no Manifesto Ágil, na década de 1990 os signatários Mike Beedle, Ken Schwaber e Jeff Sutherland desenvolveram o método que ficou conhecido como Scrum. A partir de um artigo de Nonaka e Takeuchi (1986) no qual eram destacadas as vantagens de pequenos times multifuncionais na obtenção de resultados, Jeff Sutherland, Mike Beedle e Ken Schwaber criaram em 1993 a metodologia de desenvolvimento de produtos chamada Scrum (CARVALHO, 2009). Segundo Sutherland e Schwaber (2007), o primeiro desenvolvimento com o Scrum ocorreu na Easel Corporation em 1993 e, em 1995, a metodologia foi formalizada.

O termo Scrum é a denominação da reunião realizada entre os jogadores no início de um lance do jogo *Rugby*, surgindo da comparação entre desenvolvedores e jogadores. A analogia se deu porque no jogo *Rugby* os times agem como uma unidade integrada, onde todos os membros têm o seu papel especifico e se ajudam em busca de atingir o objetivo comum. Seguindo a premissa dos times de jogadores do *Rugby*, os times de desenvolvedores deverão ter as mesmas características, habilidades e cultura ao adotar o método Scrum.

3.1. DEFINIÇÃO

Segundo Rafael Sabbagh (2013), Scrum é um *framework* Ágil, simples e leve, utilizado para a gestão do desenvolvimento de produtos complexos imersos em ambientes complexos. Scrum é embasado no empirismo e utiliza uma abordagem iterativa e incremental para entregar valor com frequência e, assim, reduzir os riscos do projeto. O Scrum aplica no desenvolvimento de software algumas ideias como flexibilidade, adaptabilidade e produtividade embasadas da teoria de controle de processos industriais.

Em suas premissas temos que o desenvolvimento de software é muito complexo e imprevisível para ser planejado totalmente no início do projeto. Ao invés disso, o processo deve ser controlado empiricamente para garantir a visibilidade, inspeção e adaptação (MARÇAL; FREITAS; SOARES; MACIEL; BELCHIOR, 2007). Baseada em uma teoria de controle empírica, o Scrum se desenvolveu como uma abordagem iterativa, incremental de otimização da previsibilidade e controle de riscos. E três pilares sustentam essa metodologia (SCHWABER, 2009):

- **Transparência:** garante que todos os aspectos relevantes ao sucesso do processo se mantenham visíveis e conhecidos de modo a garantir que o resultado obtido seja coerente ao definido previamente;
- **Inspeção:** é feita com finalidade de se detectar qualquer não conformidade que possa vir a prejudicar os resultados da equipe;
- **Adaptação:** a partir da identificação da irregularidade são feitas adaptações no processo ou no material em processo, reduzindo a probabilidade de um resultado insatisfatório.

A metodologia tem como ideia principal que no decorrer do desenvolvimento do *software* poderá ocorrer muitas mudanças nos requisitos, recursos e tecnologias que podem mudar durante o processo. Tornando assim o processo de desenvolvimento imprevisível e complexo, que acaba requerendo maior flexibilidade para acompanhar as mudanças. Segundo (FERREIRA, 2005), as principais características do SCRUM são:

- é um processo ágil para gerenciar e controlar o desenvolvimento de projetos;
- é um *wrapper* para outras práticas de engenharia de software;
- é um processo que controla o caos resultante de necessidades e interesses conflitantes;
- é uma forma de aumentar a comunicação e maximizar a cooperação;
- é uma forma de detectar e remover qualquer impedimento que atrapalhe o desenvolvimento de um produto;
- é escalável desde projetos pequenos até grandes projetos em toda empresa.

Segundo Schwaber e Beedle (2002), o Scrum tem como objetivo definir um processo de desenvolvimento de projetos focado nas pessoas da equipe. A metodologia é indicada para equipes pequenas, requisitos pouco estáveis e pequenas interações que possibilitam a visibilidade para o desenvolvimento. Assim, o Scrum exige poucos artefatos, pois não é dirigido por eles e consequentemente não requerer extensas documentações de requisitos, especificações, modelagem e diagramas. Os papeis e fases bem definidas dos métodos veremos com maiores detalhes a seguir.

3.2. PROCESSO E FUNCIONAMENTO SCRUM

O processo de desenvolvimento do Scrum se baseia em interações incrementais com duração de duas a seis semanas, chamadas de *Sprint*. *Sprint* é nada mais, nada menos que o período de trabalho para cada fase e tem seu objetivo claro e definido, conhecido por toda a equipe.

A etapa inicial dentro do Sprint é a *Sprint Planning,* onde o *Scrum Team* e o *Product Owner* definem de acordo com o *Sprint Backlog* o que será desenvolvido na interação. A etapa seguinte é a de execução, onde o *Scrum Team* de acordo com a solicitação do cliente detalha as tarefas necessárias para realizar a solicitação. Durante o Sprint é realizado as *Daily Meeting* com a duração de 15 minutos para acompanhar o progresso do desenvolvimento e informar o *Scrum Master* de eventuais impedimentos no trabalho. No fim do Sprint é realizado a *Sprint Review*, onde o *Scrum Team* apresenta o que foi desenvolvido durante o Sprint e o *Product Owner* verifica se o objetivo do Sprint foi atingido. Logo após é realizados o *Sprint Retrospective* onde ocorre a avaliação dos acertos e os erros ocorrido durante o Sprint, com o objetivo de melhorar o processo de trabalho.

A Figura 1 apresenta o fluxo do processo de desenvolvimento utilizando o Scrum:

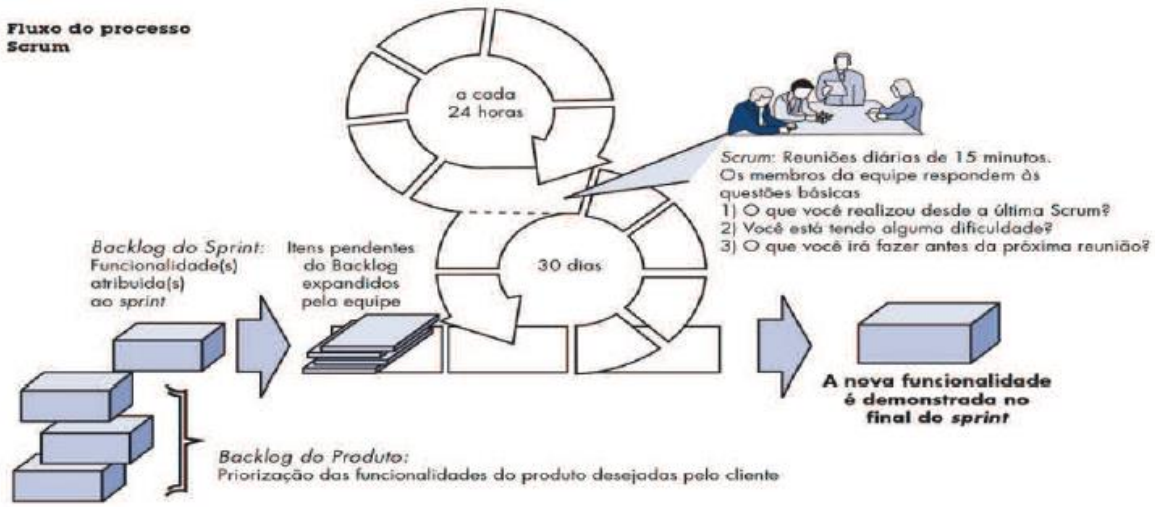


Figura 1- Fluxo do Processo Scrum

Fonte: Pressman, 2011

3.3. FASES

Segundo Larman (2004), o ciclo de vida do Scrum possui quatro fases:

- Planejamento: nesta fase a visão do projeto deve ser estabelecida e ficar clara para todos os envolvidos. As versões iniciais do *Product Backlog*, do plano de entregas e a arquitetura de negócio também são criadas.
- Preparação: nesta fase, o escopo e as dimensões do projeto ficam mais claros e, portanto, são adicionados novos itens ao *Product Backlog*. Os times são formados e mecanismos de comunicação e coordenação entre eles são definidos. Esta fase também é conhecida como *Iteration* Zero ou *Sprint* Zero, que é tratada como uma *Sprint*, porém possui um comportamento diferente das demais, pois seu objetivo não é entregar algo funcional no fim, mas sim construir a base necessária no time para que a entrega de funcionalidades seja possível (Schuh, 2005).
- Desenvolvimento: nesta fase, ocorre o desenvolvimento iterativo e incremental do produto via *Sprints*.
- Entrega: nesta fase, a entrega final do produto é realizada.

3.4. PAPÉIS

3.4.1. SCRUM MASTER

Segundo Rafael Sabbagh (2014) o *Scrum Master* trabalha para facilitar e potencializar o trabalho do Time de *Scrum*. Ou seja, utilizando-se de seu conhecimento de *Scrum*, habilidade de lidar com pessoas, técnicas de facilitação e outras técnicas, o *Scrum Master* ajuda o *Product Owner* e *Srum Time* a serem mais eficientes na realização do seu trabalho.

Rafael Sabbagh complementa, para realizar esse trabalho, o *Scrum Master*:

- Facilita o trabalho do Time de *Scrum*, de forma que seus membros se autoorganizem para que juntos desenvolvam o produto, comuniquem-se efetivamente e busquem continuamente melhorar seus processos de trabalho, realizando-o com qualidade e produtividade; além de facilitar os eventos do *Scrum*;
- Remove ou gerencia a remoção dos impedimentos que atrapalham o trabalho do Time de Desenvolvimento e ajuda a prevenir que os impedimentos aconteçam, quando possível;

- Promove as mudanças organizacionais necessárias para que o Time de *Scrum* possa realizar seu trabalho com efetividade;
- Assegura que o *Scrum* seja compreendido e adequadamente utilizado pelo Time de *Scrum*;

Na maioria das vezes o papel de *Scrum Master* é assumido pelo gerente de projeto ou líder técnico, mas pode ser assumido por qualquer membro da equipe com experiência o suficiente na metodologia, exceto o dono do produto.

3.4.2. SCRUM TEAM

O *Scrum* Team é a equipe de desenvolvimento, composta de 6 a 10 pessoas auto-organizáveis, auto-gerenciáveis e multifuncionais. Na equipe não existe necessariamente uma divisão funcional de papeis, todos devem englobar todas as características necessárias para a implementação do projeto que se propões. Todos os membros da equipe são responsáveis por desenvolver o projeto e completar o conjunto de trabalhos do *Sprint*. Com isto, induz os membros da equipe aprimorar a flexibilidade, criatividade e produtividade individual e grupal.

Segundo Ken Schwaber (Schwaber, 2004), grandes projetos que requerem mais de um time são um problema que pode ser tratado a partir da formação de um *Team of Team Member's*. O *Team of Team Member's* é composto de um *Team* que é responsável por outro *Team*, e normalmente cada equipe contribui com um membro da equipe que frequentará o *Scrum of Scrums Meeting para* coordenar o trabalho de múltiplas equipes. A Figura 2 mostra sua formação. Os encontros geralmente são realizados no *Daily Scrum*, mas não acontecem necessariamente todos os dias.

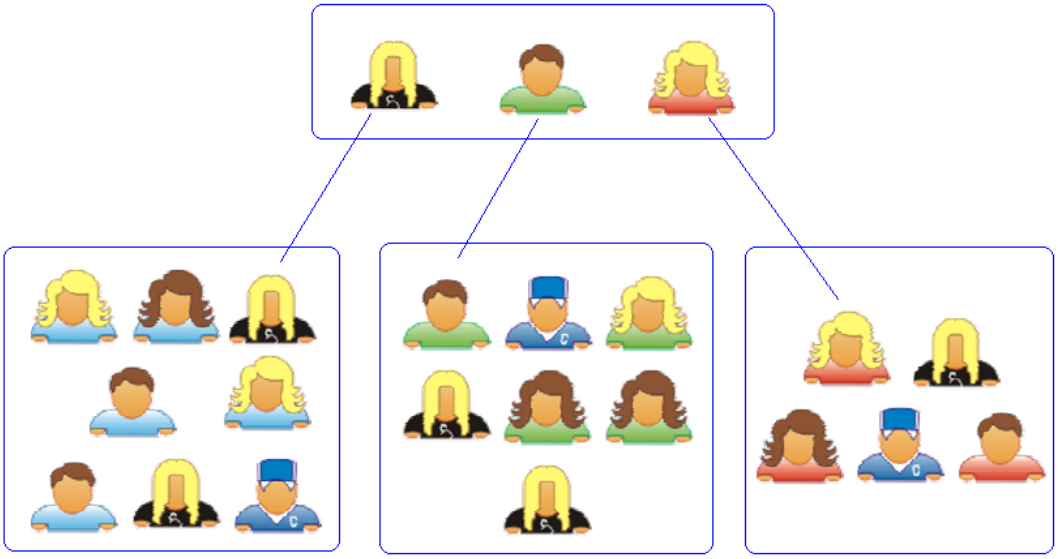


Figura 2- Scrum of Scrums

3.4.3. PRODUCT OWNER (PO)

O *Product Owner* é o responsável por representar os interesses de todos os envolvidos no projeto e também de apresentar o seu resultado (Schwaber, 2004), além de ser responsável por priorizar o *Product Backlog*. O *Product Owner* normalmente é alguém da área de *marketing* ou alguém importante no contexto do produto que simulará o papel de cliente. Entre as tarefas realizadas tem-se: definir e priorizar as atividades do *Product Backlog* e aceitar ou rejeitar as entregas do *Scrum Time*.

3.5. ARTEFATOS

Nesta seção veremos alguns dos mais importantes artefatos utilizados dentro do processo de desenvolvimento de *software* com *Scrum*.

3.5.1. PRODUCT BACKLOG

O *Product Backlog* é uma lista dinâmica e priorizada contendo tudo o que se acredita que será realizado pelo time de desenvolvimento durante o projeto. A lista contém requisitos funcionais, não funcionais, características, tecnologias, melhorias e eliminação de *bugs*. Cada item da lista tem um valor associado, onde de acordo com o seu retorno ao projeto é realizado a priorização. Inicialmente, o *Product Baklog* pode ser considerado incompleto, pois representa uma primeira lista do que o produto

necessita para atender as necessidades de mercado, porém como as necessidades mudam no decorrer do desenvolvimento, altera-se também o *Backlog* do Produto de modo a adequá-lo as exigências (SCHWABER e BEEDLE, 2002).

Segundo Schwaber (1987), para a definição dos requisitos do *Product Backlog* no desenvolvimento de um software são levadas em consideração seis variáveis:

1) Requisitos do cliente: o que deve ser melhorado no atual sistema para atender aos clientes?

2) Tempo: qual é o tempo necessário para que o produto desenvolvido tenha vantagem competitiva?

3) Competidores: onde está a concorrência e o que é preciso para que o produto desenvolvido os supere?

4) Qualidade: quais são as qualidades exigidas?

5) Visão: o que é preciso mudar e adaptar no sistema para que o desenvolvimento atinja a visão?

6) Recurso: o que existe disponível de equipe e recursos financeiros para o desenvolvimento?

O *Product Backlog* é essencial para a reunião de planejamento, é por meio dele que o *Product Owner* apresentará à equipe de desenvolvimento as principais necessidades para as *sprints*. A manutenção é geralmente feito pelo *Product Owner*, que é uma única pessoa.

3.5.2. SPRINT BACKLOG

É uma lista de tarefas, onde temos o trabalho da equipe em cada Sprint do processo. A lista nasce durante o planejamento do Sprint. As tarefas do *Sprint Backlog* são o que a equipe definiu como sendo necessário para a fluência da realização dos itens do *Product Backlog* nas funcionalidades do sistema. Cada tarefa é identificada pelo seu responsável e a sua quantidade estimada de trabalho restante (SCRUM FOR TEAM SYSTEM, 2007).

O *Sprint Backlog* é atualizado constantemente pelo *Scrum* Master para apresentar as tarefas que foram completadas e o tempo que a equipe acredita que completará as tarefas que ainda não foram finalizadas. A atualização é realizada diariamente e colocado em um gráfico, resultando assim em um *Sprint Burn down Chart*.

3.5.3. BURNDOWN CHART

O *Release Burndown chart* é um poderoso indicador visual que mostra a velocidade em que um time está se aproximando de seus objetivos (Cohn, 2005). Segundo Rafael Sabbagh (2014) o gráfico de *release Burndown* não é parte do *framework* Scrum, mas pode ser útil quando se utiliza um Plano de *Release*, geralmente estabelecido em uma reunião de *Release Planning*.

O release *Burndown chart* é representado por dois gráficos que monitoram o progresso do projeto: *Product Burndown Chart* e *Sprint Burndown Chart.* Os gráficos representam a correlação entre a quantidade de pontos que ainda faltam ser feitos e o progresso do projeto realizado para reduzir o trabalho, possibilitando assim fazer um melhor planejamento das interações.

No eixo X do *Release Burndown chart* mostra o número de *story points* da *release* e o eixo Y representa o tamanho da *release*. Como o gráfico é possível verificar o andamento do *Sprint* e se está dentro do esperado. Como pode ser visto na Figura 3, se o número de tarefas diárias não for cumpridas a linha azul fica acima da linha vermelha, com isto distanciando-se da reta de atividades diárias. Se a linha azul estiver abaixo da linha vermelha a tarefa foi superestimata e o *Sprint* deverá acabar antes. O acompanhamento faz grande diferença para o time conseguir realizar as entregas nos prazos.

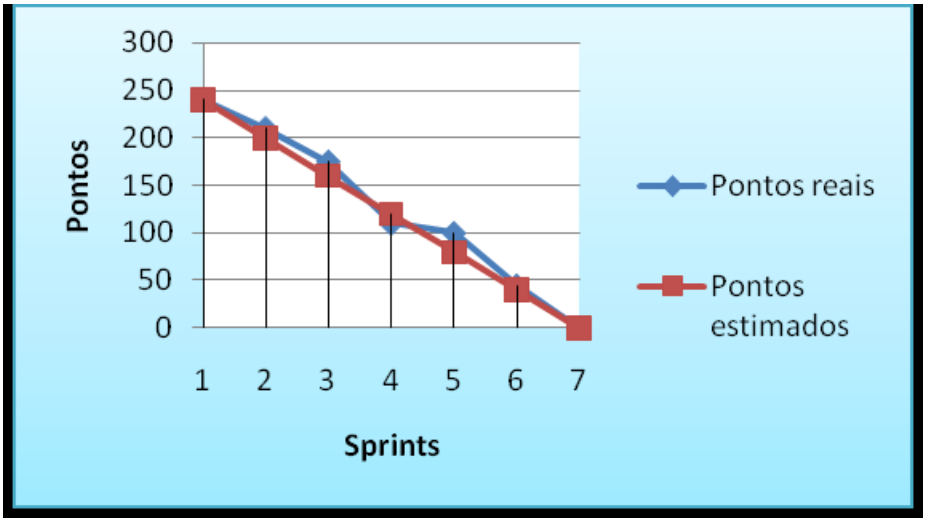


Figura 3 - Exemplo de um Release Burndown chart

3.6. CERIMÔNIAS

3.6.1. SPRINT PLANNING

O S*print Planning* é uma reunião de planejamento de *Sprint*, nela está presente o *Product Owner*, o *Scrum* Master e o *Scrum Team*. Segundo Schwaber (2004) é nesta reunião que é determinado os objetivos e funcionalidades da próxima *Sprint* e o a equipe então planeja as tarefas individuais que precisam ser feitas para criar o incremento do produto.

O *Sprint Planning* deve ter a duração exata de 8 horas para uma *Sprint* de um mês, e duração proporcional para *Sprint* com menor duração. Ela é dividida em duas partes, cada parte deverá ocorrer no prazo de 4 horas.

Parte 1: de acordo com a *Scrum Alliance* nas primeiras 4 horas o PO apresenta os itens de maior prioridade do *Product Backlog* para a equipe. O *Scrum Team* faz perguntas para entender melhor as funcionalidades e ser capaz de quebrar as funcionalidades em tarefas técnicas que irão dar origem ao *Sprint Backlog* e então definem colaborativamente o que poderá entrar no desenvolvimento do próximo *Sprint*, considerando o tamanho da equipe, a quantidade de horas disponíveis e a produtividade do *Scrum Team*.

Parte 2: Segundo a *Scrum Alliance*, durante as próximas 4 horas o *Scrum Team* planeja seu trabalho, definindo o *Sprint Backlog*, que são as tarefas necessárias para implementar as funcionalidades selecionadas no *Product Backlog*.

3.6.2. DAILY SCRUM

O *Daily* Scrum é uma reunião diária que tem como objetivo disseminar informações sobre o estado do projeto e de certa forma criar uma pressão para a entrega do objetivo. É uma reunião onde os *Scrum Team* assumem compromissos perante os demais, não caracterizando uma reunião de *status* na qual um chefe fica buscando informações dos membros que estão atrasados.

A reunião tem como principais características:

- Geralmente é realizada na parte da manhã, pois ajudará o *Scrum Team* estabelecer as prioridades do dia de trabalho que se inicia.
- Deve ter duração de 15 minutos.
- Geralmente é realizado no mesmo lugar e na mesma hora do dia.
- Os membros do time permanecem em pé.

- Geralmente deve iniciar no horário marcado, frequentemente o time define penalidade por atraso.

Os membros do *Scrum Team* respondem três perguntas:

1. O que você fez desde a última *Daily Scrum*?
2. O que você planeja fazer até a próxima *Daily Scrum*?
3. Quais são os impedimentos para o seu trabalho?

Estes impedimentos são registrados em uma lista, conhecida como *Impediments Backlog*, onde o *Scrum* Master é responsável pela priorização e gerência dos itens da lista, além de assegurar que os impedimentos sejam solucionados e removidos do *Impediments Backlog* (Sutherland, 2006).

3.6.3. SPRINT REVIEW

Realizada no fim de cada Sprint pelo *Scrum Team*, o *Scrum Master* e o *Product Owner. O Sprint Review* serve para apresentar o que foi desenvolvido durante a Sprint. Durante o *Sprint Review* o *Scrum Team* apresentará as histórias que foram desenvolvidas, assim como os problemas encontrados durante o Sprint, o *Product Owner* realizará os testes das histórias. Caso alguma história não for finalizada o *Team Srum* discutirá com o *Product Owner* os motivos.

3.6.4. SPRINT RETROSPECTIVE

A *Sprint Retrospective* é uma reunião com duração de 3 horas que ocorre ao no fim de um Sprint, tem como objetivo identificar o que houve de melhor, o que pode melhorar e as ações que serão tomadas para melhorar. Com o objetivo de garantir a melhoria incremental continua, a reunião visa assegurar regularidade e frequência na realização das melhorias, dando oportunidade ao *Time Scrum* de promover mudanças para corrigir problemas antes que afetem negativamente os resultados futuros. Sem a melhoria continua é grande a possibilidade dos times retornarem ao uso de velhas práticas e perderem a no desempenho já que não existem novos desafios.

4. EXTREME PROGRAMMING

A *Extreme Programming* (XP) é uma metodologia ágil para equipes pequenas e médias que desenvolvem software baseado em requisitos vagos e que se modificam rapidamente (BECK, 2004). Além de ser ideal para desenvolvimento de softwares orientados a objetos, equipes preferencialmente até 12 desenvolvedores e desenvolvimento incremental ou interativo.

Criada em 1996 por Kent Beck, Ron Jeffries e Ward Cuningham a partir de um projeto piloto realizado na Daymler-Crysler. O sistema em questão, conhecido como C3, ou *Chrysler Comprehensive Compensation System* (Sistema de Compensação Abrangente da Chrysler), é conhecido como o berço do XP e foi onde Kent Beck utilizou pela primeira vez, em conjunto, as práticas que atualmente formam a estrutura do *Extreme Programming* (BECK & ANDRES, 2005; TELES, 2004).

O termo *Extreme Programming* surgiu da ideia de qual seria a configuração que permitiria bons resultados se fosse possível o desenvolvimento de um sistema por meio de um painel, onde houvesse botões giratórios, e os botões representassem a importância de uma atividade no desenvolvimento do projeto. Segundo BECK (2000), a principal ideia de XP é girar todos estes os botões para o seu valor máximo. O resultado é um conjunto de práticas que, ao serem usados em conjunto, produzem um resultado estável, previsível e flexível.

A metodologia causou grande polemica por ser diferente e até mesmo conflitante em relação ao Processo Unificado, principalmente por acreditar que “o custo da mudança não deve aumentar dramaticamente com o tempo”, contrariando assim os princípios da engenharia de software. Entre os métodos ágeis, XP se destaca por ter *feedback* constante, abordagem incremental, encorajamento de comunicação face a face, flexibilidade na implantação de funcionalidades respondendo as necessidades e dependência dos testes automatizados para monitorar o progresso do desenvolvimento.

A XP concentra os esforços da equipe de desenvolvimento em atividades que geram resultados rapidamente na forma de software intensamente testado e alinhado às necessidades de seus usuários. Além disso, simplifica e organiza o trabalho combinando técnicas comprovadamente eficazes e eliminando atividades redundantes. Por fim, reduz o risco dos projetos desenvolvendo software de forma iterativa e reavaliando permanentemente as prioridades dos usuários (TELES, 2004).

Falando mais especificamente sobre o método, segundo os autores a metodologia é código-cêntrica, ou seja, o desenvolvimento é voltado para o código. Os códigos em XP são escritos em duplas, tendo em vista aumentar a qualidade do código por baixo custo. Os códigos deverão estar padronizados para que seja possível que os desenvolvedores e até mesmo os cliente validem e certifique se os requisitos estão sendo atendidos.

A adoção de XP requer muita disciplina. Para conseguir aproveitar todas as vantagens é preciso adotar todas as práticas de forma disciplinada. Enfatizando assim o trabalho em equipe como chave para um processo mais ágil e com melhor qualidade. A equipe deve ser pró-ativa, para garantir a alta produtividade, e o cliente deve estar presente para auxiliar a sanar dúvidas e tomar decisões em relação ao projeto.

A metodologia traz inúmeros benefícios, tornando o processo de desenvolvimento mais ágil e flexível. Além da agilidade no planejamento, a falta de uma especificação completa e formal dos requisitos e o desenvolvimento de sistemas simples que atendam aos atuais requisitos. XP aposta que é melhor fazer algo simples para atender de imediato e amanhã pagar mais para fazer modificações necessárias, garantindo que o que foi entregue hoje será usado. Sendo uma maneira disciplinada de desenvolver *software* existe praticas determinadas que devem ser executadas. Estas práticas e os princípios serão descritas nas sessões a seguir.

4.1. VALORES

4.1.1. COMUNICAÇÃO

A comunicação é indispensável para o sucesso do projeto de desenvolvimento de *software*. Ela está presente em todas as etapas do desenvolvimento, sendo entre os membros da equipe ou entre eles e os clientes.

Para XP a comunicação face-a-face é a forma ideal, pois é possível interpretar as ideias levando em consideração o emocional da fala, os gestos, expressões faciais, etc. É recomendado evitar ao máximo os contatos por telefone, e-mail, etc. Acreditando que a interação direta de certo modo diminui as falhas de comunicação e evitar o retrabalho desnecessário.

Segundo Cockburn (2002), a rapidez na comunicação é um aspecto relevante, pois o ritmo de progresso de um projeto está ligado ao tempo que se leva para transmitir uma informação da mente de uma pessoa para outra. Ele ainda

complementa que os custos de um projeto crescem na proporção do tempo necessário para as pessoas se compreenderem.

4.1.2. FEEDBACK

Feedback, basicamente, é a realimentação que o cliente fornece à equipe de desenvolvimento durante o projeto. O *feedback* constante possibilita que a equipe de desenvolvimento e o cliente avaliem e aprendam sobre os requisitos do sistema. Este processo não existe apenas no XP. Ele existe no desenvolvimento tradicional, mas com uma grande diferença em relação aos tempos de realização, pois no desenvolvimento tradicional é normal existir uma grande defasagem entre os ciclos de produção e consumo (TELES, 2004).

XP é organizado em curtos ciclos de *feedback*, possibilitando ao cliente solicitar novas funcionalidades, usufruir e avaliar as que foram entregues através do software em produção. Quanto antes for liberado novas versões do sistema e o cliente fornecer a sua avaliação, o projeto poderá sair mais barato e atender as reais necessidades. Com isto, a filosofia da XP se baseia no *feedback* em ciclos curtos, garantido que durante o ciclo pouco trabalho seja realizado. Com isto, facilita a localização e correção de possíveis falhas que possam vim ser encontradas, pois a funcionalidade estará englobada em um escopo pequeno e simples.

4.1.3. SIMPLICIDADE

A XP prega que deve ser desenvolvido apenas o que realmente atenderá a necessidade do cliente no momento, evitando preocupar com futuras necessidades que possam vim a surgir. Podendo na maioria das vezes encarecer o projeto e não ser usados futuramente. O que for desenvolvido deve ser simples e codificado apenas o necessário para atender o objetivo do cliente, tornando o projeto ágil e maleável. Não se deve tentar prever o futuro, pois raramente obter-se-á êxito nas previsões (TELES, 2004).

O valor simplicidade evita um erro frequente, quando o projeto é executado seguindo a premissas que não se tem total certeza. Acredita-se na maioria das vezes que é melhor adiar decisões, pois futuramente as informações possam ser confiáveis para ser embasadas. O principal objetivo da simplicidade é evitar o trabalho extra que resulta do desconhecimento ou da precipitação.

A comunicação e a simplicidades possuem um relacionamento mutuo, quanto maior a comunicação, mais claro são as necessidades dos clientes. Quanto mais simples o sistema, menor comunicação ele requer e quanto mais se comunicar, mais simples o sistema se apresentará.

4.1.4. CORAGEM

Para manter a comunicação, *feedback* e simplicidade é preciso coragem. A coragem da suporta a simplicidade, pois é possível experimentar a oportunidade de simplificar o software, além de precisar de coragem para obter juntamente com o cliente *feedback* constante. Às vezes a coragem manifesta-se como uma propensão a uma ação; se você souber qual é o problema, faça algo sobre ele. Às vezes a coragem manifesta-se como paciência; se você sabe que há um problema, mas não sabe a solução, é preciso coragem para esperar o problema tornar-se claro (BECK, 2005).

Geralmente pessoas envolvidas no desenvolvimento de software sentem medo, prejudicando as vezes na qualidade do trabalho. Para diminuir riscos e encorajar o trabalho recomenda-se sempre considerar todos os valores nas tomadas de decisão, pois é quando o medo está presente. A falta de coragem é frequente em momentos de implantação de melhorias em partes do sistema que está funcionando, realizar grandes mudanças no decorrer do projeto, tentativa de coisas pouco prováveis e até mesmo retornar ao ponto onde o sistema estava em casos de erro.

4.2. PRÁTICAS

4.2.1. CLIENTE PRESENTE

O XP sugere que o cliente esteja no dia-a-dia do projeto, acompanhando os passos dos desenvolvedores, onde a sua ausência representa sérios riscos ao projeto (*KUHN; PAMPLONA, 2007*). A participação ativa do cliente é indispensável para obter o máximo de valor do projeto, viabilizar a simplicidade do processo em vários aspectos, sanar as dúvidas e tomada imediata de decisão. Embora envolver o cliente na equipe seja positivo para o projeto, nem sempre é possível e fácil a participação do cliente como parte integrante da equipe de desenvolvimento.

4.2.2. JOGO DO PLANEJAMENTO

O jogo do planejamento consiste em uma reunião, onde as funcionalidades são avaliadas pelo cliente e priorizadas para a próxima *release* ou interação. O XP tem *releases* que tomam alguns poucos meses, que se dividem em iterações de duas semanas, que se dividem em tarefas que tomam alguns dias (BECK & FOWLER, 2001). O objetivo desta pratica é fazer inicialmente um plano aproximado e refiná-lo no desenrolar do projeto. No fim do planejamento são gerados como artefatos uma pilha de cartões contendo estórias do cliente. Para o cliente conhecer o custo da implementação de cada estória as mesmas são estimadas utilizando a unidade especial conhecida como ponto pelo time de desenvolvimento. A estimativa das estórias é importante para o cliente priorizar de maneira adequada.

Em XP as responsabilidades são divididas em técnicas e gerencias. A reponsabilidade técnica fica por conta do pessoal do desenvolvimento, que basicamente é estimar o tempo das solicitações, tomada de decisões técnicas, organização do cronograma detalhado e etc. A responsabilidade gerencial fica para o cliente, que deverá tomar decisões gerenciais, escopo das funcionalidades, priorização das estórias, compor um *release* e definir datas importantes para o projeto.

4.2.3. STAND UP MEETING

Stand up meeting é uma reunião diária que procura alinhar os membros da equipe, apresentar o trabalho executado no dia anterior e priorizar as atividades do dia que se inicia. Esses tipos de reunião são realizadas diariamente, com duração de aproximadamente 20 minutos, onde todos os membros da equipe permanecem de pé. O objetivo do *stand up meeting* é comunicar problemas e não resolvê-los (BECK & FOWLER, 2001).

4.2.4. PROGRAMAÇÃO EM PAR

O XP exibe que todo e qualquer código implementado no projeto seja efetuado em dupla, chamada de programação em par. É uma técnica onde dois programadores trabalham juntos no mesmo problema, ao mesmo tempo e no mesmo computador. Um deles é responsável pela digitação (condutor) e outro acompanha o trabalho do parceiro (navegador) (KUHN; PAMPLONA, 2007). Dentre as vantagens que o estilo de programação tem é a possibilidade de aprendizagem continua entre os

programadores, redução dos riscos de eventuais falhas, implementação mais simples e eficaz e encorajamento da comunicação entre a equipe.

Com a programação em par é realizada constantemente a inspeção do código pelo outro programador. Tornando assim o processo de inspeção parte natural do dia a dia da equipe, acelerando a depuração do código. A programação em par permite as equipes superarem melhor a entrada ou saída de um programador chave no projeto. Segundo Williams & Kessler (2003), com a programação em par, o risco do projeto associado à perda de um programador chave é reduzido porque existem várias pessoas familiarizadas com cada parte do sistema.

Costuma estar presente durante a programação em par o efeito pressão por par, fazendo com que os programadores se concentrem mais e melhor, pois a responsabilidade é compartilhada. Com o seu parceiro olhando, contudo, existem boas chances de que mesmo que você esteja ignorando uma destas práticas, o seu parceiro não estará. (...) as chances de ignorar o seu compromisso com o restante da equipe são bem menores trabalhando em par que se você estivesse trabalhando sozinho (BECK, 2000).

4.2.5. DESENVOLVIMENTO GUIADO PELOS TESTES

Segundo Beck (2000), a técnica conhecida como desenvolvimento guiado pelos testes consiste em um mecanismo de teste automatizado antes de codificar cada história e cada método do sistema. Weinberg (1971) complementa, trata-se de uma técnica preventiva utilizada durante todo o projeto e a manutenção. A prevenção é usada porque antes da questão de como resolver problemas vem a questão de evitar problemas. Ele afirma que um programador que evita problemas é mais “inteligente” que aquele que traz problemas para sim mesmo. Entre as suas vantagens tem melhor projeto, melhor documentação, menos erros, menores custos, melhor aprendizado e maior produtividade.

XP concentra-se sobretudo em dois tipos de teste: o teste de unidade e o teste de aceitação. O primeiro tenta assegurar que cada componente do sistema funcione corretamente. O segundo procura testar a interação entre os componentes, as quais dão origem às funcionalidades (BECK, 2000).

O teste de unidade são testes automatizados, escritos antes de cada método ser produzido. Toda vez que o código é escrito o programador deverá verificar se o sistema passa em todos os testes, e só deverá ser integrado se passar pelos teste.

Esta abordagem proporciona códigos simples e funcionais, visão clara da finalidade do código e um conjunto de teste completo.

O teste de aceitação “são escritos para assegurar que o sistema como um todo funciona. (...) Eles tipicamente tratam o sistema inteiro como uma caixa preta tanto quanto possível” (FOWLER, 2000). Estes testes são escritos pela cliente ou com a sua orientação, pois são usados para verificar o que o sistema faz como um todo e dar *feedback* aos programadores e cliente sobre os *bugs* encontrados.

4.2.6. REFATORAÇÃO

A refatoração é o processo de alterar o código existente sem afetar a funcionalidade implementada por ele, alterando assim o como ele faz e não o que ele faz. O objetivo é aprimorar a estrutura interna (ASTELS, 2003), além de melhorar qualidades como simplicidade, flexibilidade, clareza e desempenho. Segundo POPPENDIECK & POPPENDIECK (2003) sem melhoria contínua, qualquer sistema de software irá sofrer. As estruturas internas irão se calcificar e se tornar frágeis.

A adoção da refatoração faz com que ao longo do tempo seja mantida no sistema a simplicidade, clareza, adequação ao uso, ausência de repetição e ausência de funcionalidades extras. Uma forma disciplinada de limpar o código minimizando a chance de introduzir *bugs* (FOWLER, 2000). A refatoração se torna necessária quando a arquitetura é evoluída, amadurece e é solicitado novas funcionalidades pelos usuários.

4.2.7. PROPRIEDADE COLETIVA

Em XP o código não deve ser segmentado em partes, de modo que cada programador fique responsável por uma delas. Os programadores têm acesso a todas as partes do código e tem autonomia para alterar aquilo que julgar importante e necessário, sem solicitar autorização. O código é aberto a todos os programadores, pois o código é coletivo. Com isto, todos são responsáveis pelo código, evitando assim o caos de falta de propriedade. A propriedade coletiva fornece maior agilidade, pois não é preciso esperar para o programador responsável para realizar a alteração necessária. Criando assim mais um mecanismo de revisão e verificação do código. Para a adoção eficaz da propriedade coletiva é preciso que a equipe invista em testes automatizados.

4.2.8. CÓDIGO PADRONIZADO

Os projetos desenvolvidos seguindo o XP, para que o código fique mais fácil de entender, maior consistência no código, simplificar a comunicação, desenvolver e realizar manutenção os programadores definem uma padronização para codificação. Padrões de código (...) são importantes porque levam a uma maior consistência dentro do seu código e o código de seus colegas de equipe. Mais consistência leva a um código mais simples de compreender, o que por sua vez significa que é mais simples desenvolver e manter (AMBLER, 2000).

A adoção de código padronizado faz com que a equipe se distraia com discussões com coisas de menor importância e suportar as outas coisas. A padronização permite que não seja identificado quem escreveu determinada parte do código, parecendo que o código foi editado pela mesma pessoa. O padrão deve começar simples e evoluir conforme a equipe venha adquirir experiência, com isto fica mais fácil para colocar em pratica.

4.2.9. DESIGN SIMPLES

Simplicidade é um princípio do XP, as equipes procuram garantir que os códigos sejam desenvolvidos com projetos simples, flexíveis e suficientes para atender as necessidades da funcionalidade que está implementado. A premissa adotada em XP é que as necessidades futuras sejam desenvolvidas quando ela surgir. Segundo Beck(2000), um projetos complicado é mais difícil de comunicar que um simples. Devemos, portanto, criar uma estratégia de projeto que gere o projeto mais simples possível, consistente com nossos demais objetivos.

Os sistemas devem ser capazes de se adaptar a mudanças tanto de negócio, quanto técnicas de forma econômica. Um processo de projeto tolerante a mudanças tem mais chances de criar como resultado um sistema tolerante a mudanças (POPPENDIECK & POPPENDIECK, 2003).

4.2.10. METÁFORA

Em XP, metáfora é adotada como uma analogia entre o software e um sistema, não necessariamente de software. Com o objetivo de manter a integridade conceitual de um sistema, tornando-o mais fácil de ser utilizado e entendido por clientes e programadores. Com isto possibilita com que as pessoas entendam de uma forma mais rápida os problemas encontrados no sistema, facilitando assim a comunicação

e fixação dos assuntos. Uma metáfora compartilhada apropriada permite que uma pessoa suponha precisamente onde outra pessoa da equipe adicionou código e como se encaixar com o código dela (COCKBURN, 2002).

4.2.11. RITMO SUSTENTÁVEL

O XP adota a prática de ritmo sustentável, BECK & FOWLER (2001) em síntese ela recomenda que os membros da equipe de desenvolvimento trabalhem apenas durante o tempo regulamentar, ou seja, oito horas por dia, e evitem horas-extras tanto quanto possível. Além disso, sugere-se que os prazos sejam mantidos fixos e as cargas de trabalho sejam ajustadas (via priorização) para se adequar aos prazos.

A adoção da carga de trabalho superior a 40 horas semanais, poderá acarretar no aumento do cansaço, insatisfação no trabalho, queda da qualidade do código, maior rotatividade e etc. Caso precise aumentar a carga de trabalho por duas semanas consecutivas, provavelmente existe um problema no projeto que não pode ser resolvido com aumento da carga de trabalho e sim com melhoria no planejamento. O processo de priorização e ajuste do escopo é essencial para tratar a pressão de tempo.

4.2.12. INTEGRAÇÃO CONTÍNUA

No XP a integração continua de software significa armazenar na base unificada de código as funcionalidades desenvolvidas. A integração pode ser realizada várias vezes ao dia, mas só pode ser realizada se o código não houver erro, caso contrário os mesmos devem ser corrigidos. Segundo Beck (2000), o processo de integração é serial, isto é, somente um par pode integrar o seu código de cada vez. Isso assegura que eventuais erros de integração estarão sempre relacionados a um único par: aquele que está integrando no momento. Somente após assegurar que a integração está perfeita, todos os testes executam com sucesso e o sistema encontra-se em um estado consistente poderá outro par fazer a integração.

Um modo simples de garantir o processo serial de integração é ter uma máquina dedicada somente para isto. Quando a máquina estiver livre, um par de desenvolvedores com código a integrar, a ocupa e carrega a versão corrente do sistema, depois carrega as alterações que fizeram, resolve possíveis colisões e roda os testes até que todos eles passem. Assim todos sempre sabem quando podem ou não integrar (TELES, 2004).

4.2.13. RELEASES CURTOS

Releases são pequenas versões do sistema que são entregues frequente (dois a três meses), contendo os requisitos mais importantes para o negócio. *Releases* curtos aumenta a possibilidade de *feedback* rápido do cliente, facilita o aprendizado e a correção dos bugs encontrados nos softwares. Uma das coisas mais importantes que você pode fazer em um projeto XP é liberar *releases* cedo e com frequência. (...) Você não quer perder a chance de aprender o que os usuários realmente querem. Você não quer perder o aumento na confiança que virá quando você mostrar às pessoas que fez alguma coisa útil (JEFFRIES, ANDERSON et al., 2001).

5. CRYSTAL

Crystal é uma família de diferentes metodologias, que devem ser ajustadas de acordo com o projeto e equipe. Criada por Alistair Cockburn e Jim Highsmith. Segundo COCKBURN (2002) Crystal é caracterizada por um jogo cooperativo de invenção e comunicação de recursos ilimitados, com o principal objetivo de entregar softwares prontos e funcionando e com o objetivo secundário de preparar-se para o jogo seguinte.

Os membros da família Crystal são indexados por cores diferentes para indicar o "peso": claro, amarelo, laranja, vermelho, magenta, azul, violeta, e assim por diante. Quanto mais escura for a cor, mais "pesada" será a metodologia. Um projeto com 80 pessoas precisa de metodologias mais pesadas do que um com apenas 10 pessoas (Abrahamsson et al, 202).

O método foca nas pessoas e não nas atividades e artefatos, além focar na eficiência e habilidade da equipe. Algumas das características do método Crystal são (Cockburn, 2004):

- entrega frequente de código utilizável pelo usuário;
- melhoria contínua refletiva;
- foco – trabalhar sobre as prioridades;
- fácil acesso para usuários experientes;
- restes automatizados, gerência de configuração e integração frequente.

Além das características mencionadas por Cockburn pode destacar também desenvolvimento incremental com ciclos de no máximo quatro meses, ênfase na comunicação e cooperação das pessoas.

O ciclo de vida desta família de metodologia é baseado nas seguintes práticas:

- **Staging**: planejamento do próximo incremento;
- **Edição e revisão:** desenvolvimento, apresentação e revisão dos objetivos do incremento;
- **Monitoramento:** é realizado o monitoramento da equipe;
- **Paralelismo e fluxo:** verifica e propõe as operações em paralelismo, monitorando estabilidade entre equipes;
- **Inspeções de usuários:** a cada incremento é sugerido que os usuários realize inspeções;

- **Workshops refletivos:** são reuniões que ocorrem antes e depois de cada iteração com objetivo de analisar o progresso do projeto.
- **Work Products:** sequência de lançamento, modelos de objetos comuns, manual do usuário, casos de teste e migração de código.
- **Padrões:** definições de padrões, desde simples notações até contratos de um produto;
- **Ferramentas:** descrição das ferramentas utilizadas durante a interação.

6. MODELAGEM SCRUM

Para exemplificar o processo, utilizaremos como exemplo um novo módulo de um sistema de crediário. O sistema é especialista em concessão de crédito e fortalecimento de soluções para pagamento por meio de crediário/boleto, com o objetivo de aumentar as vendas com segurança e agilidade.

O sistema foi desenvolvido para empresas que querem ampliar seus negócios, diminuir o risco e fidelizar seus consumidores, possibilitando aumentar a oferta de crédito a consumidores com garantia de recebimento em casos de inadimplência. O sistema fornece aos clientes as seguintes vantagens:

- Aumento das vendas.
- Possibilidade de trabalhar com público com pouco acesso ao mercado de crédito.
- Maior poder de compra para o consumidor.
- Aumento da receita.
- Garantia de recebimentos.
- Terceirização de custos operacionais e gestão.
- Terceirização no atendimento ao consumidor.
- Aumento na agilidade na concessão de crédito.
- Menos burocracia no cadastro do consumidor.
- Gestão das vendas e pagamento através de relatórios gerenciais.

6.1. PAPÉIS

Para iniciar o projeto é de grande importância a definição dos papéis dos envolvidos. De acordo com o Scrum, são descriminados abaixo os membros e seus respectivos papéis no processo de desenvolvimento do projeto.

PAPEIS	
Product Owner	Carlos Eduardo
Scrum Master	Solange Zago
Scrum Team	Ericsen Lucas Dilson Lopes

6.2. ESCOPO

Com o aumento de supostos casos de fraudes no sistema de crediário, as diretorias organizacionais, juntamente com o setor de recuperação sentiram a necessidade de realizar uma análise aprofundada nas compras. A análise das compras possibilitaria constar se a compra realmente foi fraude ou se o consumidor não reconhece a dívida usando de má fé.

Após várias reuniões o Product Owner Carlos Eduardo, juntamente com o setor de recuperação da organização definiram o escopo do projeto.

O projeto apelidado como “Analise de Compra”, permitie ao setor de recuperação e ao setor jurídico, realizarem uma análise das compras que ao iniciar a fase de cobrança os consumidores alegam não reconhecer a dívida

Este novo modulo no sistema terá como principais funcionalidades:

- Permitir que o setor de recuperação e o setor jurídico realize análise aprofundadas das compras;
- Apresentar informações referente aos cadastros realizados na rede com o mesmo CPF.
- Apresentar todos os documentos digitalizados na rede para o CPF.
- Apresentar as compras realizadas na rede para o CPF.
- Apresentar o histórico das parcelas da compra, onde pode ocorrer pagamento, acatamento, perda do título.
- Apresentar a situação do consumidor, caso ele esteja negativado ou não.
- Histórico de analises da compra.
- Retirar o CPF do consumidor dos órgãos de proteção ao crédito.
- Suspender a cobrança por meio de cartas e SMS.
- Permitir que o cliente solicite o ressarcimento das parcelas da compra futuramente.
- Não permitir que o cliente solicite o ressarcimento de compras antes do prazo estabelecido.
- Permitir que o usuário cadastre os motivos da fraude.
- Apresentar nas consultas do sistema as compras que foram analisadas para o consumidor pesquisado.
- Permitir o consumidor pesquisar por Ramo e/ou Filial e/ou Cliente e/ou Tipo do Motivo e Período as compras que sofreram análise.

- Permitir o consumidor pesquisar por CPF ou Número da Compra as compras que sofreram análise.

6.3. PRODUCT BACKLOG

Após a definição do escopo do projeto, o Product Owner cria o Product Backlog para logo após realizar a priorização das funcionalidades de acordo com o valor que as mesmas agregarão no negócio. No Product Backlog abaixo é utilizado uma escada de 1 a 99 definir a prioridade, onde quanto maior o número menor prioridade a funcionalidade terá e consequentemente menor valor trata para o negócio.

Funcionalidade	Prioridade
Modelagem dos Dados	1
Cadastro de motivos de fraude	2
Análise Compra	3
Reanálise Compra	4
Solicitar Acatamento	5
Consulta por Ramo e/ou Filial e/ou Cliente e/ou Tipo do Motivo e Período	6
Consulta por CPF e Número de Venda	7
Consulta Compras	8

6.4. REUNIÃO DE PLANEJAMENTO DE SPRINT

Após a priorização das funcionalidades do projeto, é realizada a reunião de planejamento. O *Product Backlog* é apresentado para o *Scrum Team,* onde é definido a quantidade de horas que serão necessárias para desenvolver as funcionalidades, levando em consideração os aspectos técnicos.

Funcionalidade	Prioridade	Custo-horas
Modelagem dos Dados	1	8
Cadastro de motivos de fraude	2	16
Análise Compra	3	40
Reanálise Compra	4	40

Solicitar Acatamento	5	12
Consulta por Ramo e/ou Filial e/ou Cliente e/ou Tipo do Motivo e Período	6	24
Consulta por CPF e Número de Venda	7	24
Consulta Compras	8	8
TOTAL		**172**

Por meio do Product Backlog o Scrum Master, juntamente com a Scrum Team, definem o Sprint Backlog, subdividindo as tarefas grandes em pequenas tarefas.

Funcionalidade	Prioridade	Horas
Modelagem dos Dados	**1**	**8**
Definição de dados	1.1	4
Modelagem das Tabelas	1.2	3
Implementação do SGBD	1.3	1
Análise Compra	**2**	**40**
Tela para apresentar informações referente aos cadastros realizados na rede para o CPF e histórico de análises e reanálise da compra	2.1	6
Tela para apresentar todos os documentos digitalizados na rede para o CPF	2.2	6
Tela para apresentar as compras realizadas na rede para o CPF	2.3	6
Tela para apresentar presentar o histórico das parcelas da compra	2.4	6
Função para manipulação dos dados e consistência das informações.	2.5	6
Rotina para retirar o CPF do consumidor da negativação	2.6	5
Rotina para suspender a cobrança através de cartas e SMS	2.7	5

Reanálise Compra	**3**	**40**
Tela para apresentar informações referente aos cadastros realizados na rede para o CPF e histórico de análises e reanálise da compra	3.1	6
Tela para apresentar todos os documentos digitalizados na rede para o CPF	3.2	6
Tela para apresentar as compras realizadas na rede para o CPF	3.3	6
Tela para apresentar o histórico das parcelas da compra	3.4	6
Função para manipulação dos dados e consistência das informações.	3.5	6
Rotina para inserir o CPF do consumidor da negativação	3.6	5
Rotina para iniciar a cobrança através de cartas e SMS	3.7	5

6.5. INÍCIO DO SPRINT

Após a definição das tarefas, prazos e metas é possível iniciar o desenvolvimento do *Sprint*. O objetivo da Sprint é no fim apresentar um produto onde o usuário poderá cadastrar e editar os possíveis motivos da fraude, separando por análise da compra ou reanálise da compra.

Em cada Sprint é importante desenvolver um produto simples e funcional no tempo que foi determinado pela equipe. No desenvolvimento das *Sprints* o tempo deverá sempre ser ajustado a fim de não finalizar a tarefa muito rápido ou não exceder o prazo estabelecimento. O *Scrum* time deverá desenvolver durante cada ciclo os seguintes conjuntos de sub-ciclos:

- Desenvolvimento: implementação, testes e documentação;
- Revisão: o produto é revisado com o intuito de certificar se atingiu o objetivo;
- Ajustar: é realizado modificações nos requisitos ou planos.

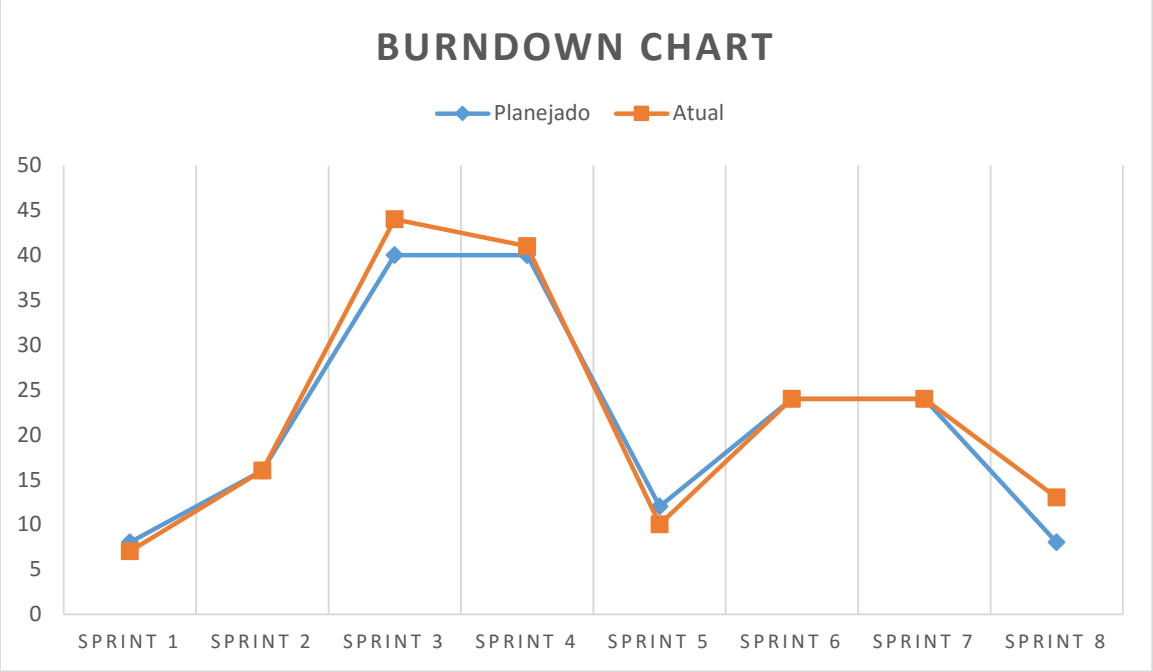
6.6. REUNIÃO DIÁRIA SCRUM

Diariamente o Scrum Team deverá reunir juntamente com o Scrum Master para realizar a Daily Scrum. O objetivo da reunião é que o Scrum Master acompanhe o desenvolvimento do projeto e aportar se o Scrum Team está comprometido com o projeto, se está completando as tarefas dentro do prazo e se encontraram alguma dificuldade.

6.7. BURNDOWN CHART

A partir das reuniões diárias, o Scrum Master construirá o Burndown Chart. O Burndown Chart permitirá monitorar o progresso do desenvolvimento do software até o fim do *Release*. O gráfico auxiliará na tomada de decisões em mudanças de rumo, assim como mudanças no escopo, meta, adiamento ou o cancelamento do *release*.

O Burddown Chart abaixo apresentará o progresso das Sprints do projeto.



Com o BurnDown Chart do projeto apresentado é notável que o desenvolvimento do projeto superou o tempo planejado. Com a velocidade média de 8 horas diárias, na Sprint 3 já seria possível reconhecer a baixa produtividade. Na Sprint 1 e 5 o Team Scrum conseguiu finalizar em um tempo menor do planejado, enquanto nos demais o tempo gasto superou o planejado em 7 horas.

6.8. REVISÃO FINAL DA SPRINT

No fim da *Sprint* todos os membros da equipe se reúnem para a apresentação dos resultados obtidos e os problemas encontrados no decorrer do desenvolvimento. O *Product Owner* identifica o progresso alcançado e juntamente com o cliente revisam o produto final. Caso identifique que falta alguma história o Scrum Team deverá discutir com o Product Owner os motivos. Com isto é finalizado a primeira Sprint. O processo é repetido novamente para todas as tarefas do Product Backlog.

7. CONSIDERAÇÕES FINAIS

Neste trabalho foi abordado algumas das metodologias ágeis mais utilizadas atualmente para o desenvolvimento de softwares: Scrum, Extreme Programming e Crystal. Com ênfases, abordagem e aplicabilidade diferentes, cada metodologia melhor se aplica para atender a necessidade de uma determinada organização. O Scrum possui como foco a gestão do desenvolvimento de produtos complexos imersos em ambientes complexos. Estreme Programming é voltado para agilizar o desenvolvimento do software, por meio de suas técnicas. Por fim Crystal é voltada para atender as equipes de diferentes tamanhos, estabelecendo princípios para cada formato de projeto tendo como base a sua complexidade. Com isto foi analisado a metodologia que melhor se aplicava ao desenvolvimento de um sistema de crédito.

Após a análise das principais metodologias ágeis pode-se concluir que o Scrum é a que melhor se aplica e satisfaz a real necessidade da organização para o desenvolvimento do software. A adoção do Scrum como metodologia de desenvolvimento ágil tende a facilitar a execução dos projetos, apresentar resultados eficaz e satisfazer o cliente final. A sua implementação é simples, tendo como desafio a integração no ambiente organizacional e que a equipe esteja receptiva a adotar uma nova cultura de desenvolvimento de software.

A implantação do Scrum na abordagem apresentada nesse trabalho, tende a auxiliar ou até mesmo assumir todo o processo de desenvolvimento, ajustando ao contexto da organização e a equipe, fazendo com que funcione e traga benefícios e cumpra os propósitos da metodologia.

Como trabalhos futuros pretendo realizar um plano de implantação da metodologia Scrum para apresentar ao gerente de projeto da organização e estudar os impactos e principais resistências do time de desenvolvimento.

8. REFERÊNCIA

BECK, K. Programação Extrema (XP) Explicada: Acolha as Mudanças, Bookman, 2004.

BECK, Kent. Extreme Programming explained: embrace change. 1. ed. Reading, MA: Addison-Wesley, 2000.

BOEHM, B.; TURNER, R. (2003). Balancing agility and discipline: a guide for the perplexed. Addison Wesley.

BOEHM, B.; TURNER R. (2005). Management challenges to implementing agile processes in traditional development organizations. IEEE Software, vol. 22, n. 5, pp. 30 - 39.

BOEHM, B.; TURNER, R. Balancing agility and discipline: a guide for the Perplexed. Boston: Addison Wesley, 2003. Disponível em: <http://ptgmedia.pearsoncmg.com/images/9780321186126/samplepages/0321186125.pdf>. Acesso em: 14 de janeiro 2016.

CARVALHO, B. V. *Aplicação do método ágil Scrum no desenvolvimento de produtos de softwares em uma empresa de base tecnológica*. Dissertação de Mestrado. Programa de Pós-Graduação em Engenharia de Produção. Universidade Federal de Itajubá/MG, 2009.

COCKBURN, Alistair. Agile Software Development. Adisson-Wesley, 2001.

COCKBURN, A.; HIGHSMITH, J. (2001). Agile Software Development: The People Factor. IEEE Computer, vol. 34, n. 11, pp. 131 - 133.

COHN, M. Agile estimating and planning. Englewood Cliffs, NJ, Estados Unidos: Prentice Hall PTR, 2005.

FAGUNDES, Priscila Bastos*. Framework para Comparação e Análise de Métodos Ágeis.* Dissertação de Mestrado. Universidade Federal de Santa Catarina, Florianópolis, 2005.

FERREIRA, D.; COSTA, F.; ALONSO, F.; ALVES, P.; NUNES, T. SCRUM - Um Modelo Ágil para Gestão de Projetos de Software. Disponível em: <http://paginas.fe.up.pt/~aaguiar/es/artigos%20finais/es_final_19.pdf>. Acesso em 22 de janeiro de 2016.

FILHO, Edes Garcia da Costa; PENTEADO, Rosângela; SILVA, Júnia Coutinho AnacletoAnderson D. J, “Agile management for software engineering, applying the theory and constraints for business results.” Prentice Hall, Upper Saddle River, NJ, 2003.

FILHO, Hélio Fernando Bentzen Pessoa. *Um estudo analítico entre as abordagens de Engenharia de Requisitos nas Metodologias Ágeis XP, SCRUM e Crystal*. 2011. Monografia – Centro de Informática – Universidade Federal de Pernambuco, Recife, 2011.

FOWLER, Martin. The New Methodology. 2005. Disponível em: <http://www.martinfowler.com/articles/newMethodology.html>. Acesso em: 14 janeiro 2016.

HIGHSMITH, HIGHSMITH, Jim. Agile Software Development Ecosystems. Adisson-Wesley, 2002.

LARMAN, Craig. Applying UML And Patterns. 2nd Edition, 2002.

MARÇAL, Ana Sofia Cysneiros; FREITAS, Bruno Celso Cunha; SOARES, Felipe Santana Furtado; MACIEL, Teresa Maria Medeiros; BELCHIOR, Arnaldo Dias. Estendendo o SCRUM segundo as Áreas de Processo de Gerenciamento de Projetos do CMMI. CLEI2007: XXXIII Conferencia Latinoamericana de Informática, San Jose, Costa Rica. Disponível em: Acesso 27 de agosto de 2007.

SABBAGH, Rafael. Scrum: Gestão Agil para Projeto de Sucesso. 1. Ed. São Paulo: Casa do Código, 2013.

SCHWABER, K.; BEEDLE, M. Agile Software Development with Scrum. New Jersey: Prentice Hall, 2002.

(Schuh, 2005) Schuh, Peter. Integrating Agile Development in the Real World. Charles River Media, 2005.

Schwaber, Ken. SCRUM Development Process. Disponível em: http://jeffsutherland.com/oopsla/schwapub.pdf. Acesso em: 01 de fevereiro de 2016.

SCHWABER, K. Agile Project Management with Scrum. Redmond: Microsoft Press, 2004.

SCHWABER, K. Scrum development process, 1987. Disponível em: <http://www.jeffsutherland.org/oopsla/schwapub.pdf >. Acesso em: 22 de janeiro 2016.

SOMMERVILLE, Ian. Engenharia de Software. 9. Ed. São Paulo: Pearson, 2011.

Larman, C. Agile & Iterative Development, A Manager's Guide. Addison-Wesley, 2004.

SCRUM for Team System. Disponível em: <https://scrumforteamsystem.codeplex.com>. Acesso em 21 de janeiro de 2016.

ZANATTA, Alexandre. xScrum: Uma Proposta de Extensão do xScrum para adequação ao CMMI. Dissertação de Mestrado. Universidade Federal de Santa Catarina. Florianópolis: 2004.

WILLIAMS, Laurie; KESSLER, Robert. Pair programming illuminated. 1. Ed. Boston, MA: Addison-Wesley, 2003. 265 p. ABRAHAMSSON, P.; WARSTA, J.; SIPONEN, M. T.; RONKAINEN, J. (2003).

SCHWABER, K. *Scrum development process*, 1987. Disponível em: < http://link.springer.com/chapter/10.1007/978-1-4471-0947-1_11#page-1 >. Acesso: 01 de feverieiro de 2016

https://www.scrumalliance.org/

SUTHERLAND, J.; SCHWABER, K. *The Scrum papers: nuts, bolts, and origin of an agile process*, 2007. Disponível em: < http://www.scruminc.com/scrumpapers.pdf />. Acesso em 28 de janeiro de 2016.

BECK, Kent; ANDRES, Cynthia. Extreme Programming explained: embrace change. 2. ed. Upper Saddle River: Addison-Wesley, 2005.

TELES, Vinicius Manhães. *Um estudo de caso da adoção das praticas e valores do extreme programming*. 2005. Dissertação – Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2005.

TELES, Vinícius Manhães. Extreme Programming: aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade. 1. Ed. São Paulo: Novatec, 2004.

TELES, Vinícius Manhães. Um Estudo de Caso da Adoção das Práticas e Valores do Extreme Programming. Rio de Janeiro, 2005. Dissertação (Mestrado em Informática) - Núcleo de Computação Eletrônica, Universidade Federal do Rio de Janeiro. Disponível em: http://www.desenvolvimentoagil.com.br/xp/dissertacaoXP.pdf
Acesso em 12 de fevereiro de 2016.

TELES, Vinicius Manhães. *Extreme Programming: Aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade*. 2. Ed. Rio de Janeiro: Novatec, 2014. 328 p.

KUHN, Giovane; PAMPLONA, Vitor. Apresentando XP. Encante seus Clientes com Extreme Programming. 2009 . Disponível em:
<http://www.javafree.org/content/view.jf?idContent=5>. Acesso em: 14 de janeiro de 2016.

BECK, Kent; FOWLER, Martin. Planning Extreme Programming. 1. ed. Boston:

Addison-Wesley, 2001.
WEINBERG, Gerald M. The psychology of computer programming. 1. ed. New York: Van Nostrand Heinhold Company, 1971. 288 p.
ASTELS, David. Test-driven development: a practical guide. 1. ed. Upper Saddle River, NJ: Prentice Hall PTR, 2003.
POPPENDIECK, Mary; POPPENDIECK, Tom. Lean software development: an agile toolkit. 1. ed. Upper Saddle River, NJ: Addison-Wesley, 2003
AMBLER, Scott W. Writing robust Java code: the AmbySoft Inc. coding standards for Java. 2000. http://www.ambysoft.com/downloads/javaCodingStandards.pdf. Acesso em 20 de janeiro de 2016
JEFFRIES, Ron; ANDERSON, Ann; HENDRICKSON, Chet. Extreme Programming installed. 1. ed. Boston: Addison-Wesley, 2001. 265 p.
SEVERO, Leonardo Souza. *Fatores críticos de sucesso na adoção de métodos ágeis*. 2014. 99. Monografia – Universidade Federal do Rio Grande do Sul, Porto Alegre, 2014.